Opracowanie:
2006l
Degrees of Separation

HISTORIA:

dokument systemu SINOL 1.6

1 Introduction

This is a classical problem of finding most distant vertices in a graph (its diameter). For simple graphs, with particular structure, for example trees, this problem can be solved more efficiently than counting distances between all pairs of vertices — we can compute diameter of that tree using dynamic programing.

2 Model solution

Model solution computes distances between each pair of vertices using Floyd–Warshall algorithm. Computational complexity of that algorithm is $O(n^3)$, it requires $O(n^2)$ memory. Implementation of this solution doesn't require lot of code lines which is its big advantage.

3 Other solutions

3.1 Optimal solutions

Another way to solve the problem in optimal time ($O(n^3)$) is to do breadth–first search for each vertex. This will give us distances between starting vertex and any other vertex in given network. This algorithm requires $O(n)$ memory. So, as it comes to memory consumption, this is quite an advantage. Nevertheless when it comes to coding, complexity of this algorithm is its big drawback (the code is rather big so its easier to make a mistake).

3.2 Suboptimal solutions

Optimal solution is quite intuitive so it's hard to say something about suboptimal solutions without trying to artificially increase computational complexity. It's possible to think about solutions, that would compute distances between each pair of vertices independently using breadth–first search (it would increase computational complexity to $O(n^4)$, or, using Dijkstra algorithm, up to $(O(n^4) \cdot \log n)$).

3.3 Incorrect solutions

Because obvious solution is also optimal solution, it's hard to point out some common mistakes. But it's possible to imagine a situation in which one would try to compute maximal distance from only one vertex (using, for example, breadth–first search) and trying to assume that diameter of the graph is maximum of computed values.

4 Remarks

First step in solution for this problem is conversion of graph representation. In the input we have names but it's easier for computer to work on numbers. Therefore we would be making on–line dictionary, which will be translating people's names into natural numbers. Algorithm for doing that is quite obvious. Given a name we try to find it in the dictionary. If search succeeds we would take number that is assigned to that name, if it fails we would assign to it following number and add to the dictionary.

There is another problem with names. In problem standing we don't have any restrictions on their lengths. It's essential to read data in a way that is independent from lengths of names, which, potentially, can be very long.

5 Tests

We've decided to create set of 10 tests. Each test consists of graphs with fixed structure. Graph can have up to $5 * test\_number$ vertices.
In the tests we distinguish subgraphs of following structure:

- paths — sequence of vertexes connected into one, long path, in this way we can get maximal degree of separation;
- cycles — similarly to paths but there are two distinct paths between two vertices, thanks to that maximal degree of separation is less or equal to half of the cycle length;
- trees — graphs that doesn't contain cycles, in that case we can get medium values of degrees of separation;
- cliques — graphs, in which each pair of vertices is connected with edge, in cliques maximal degree of separation is equal to 1;
- disconnected graphs — in that case maximal degree of separation is ∞.

Commonly graphs are filled with random edges, creating graphs with given density. Among tests there are some totally random graphs with given density, however in that case there is big probability that graph would be either disconnected or would have small degree of separation.